Sistema inteligente para la asignación de horarios a los docentes de la Tecnología en Desarrollo de Software

Julian Osorio Salazar & Nicolas Rodríguez Fernández.

2019

Universidad Tecnológica de Pereira

Ingeniería de Sistemas y Computación

Proyecto de grado

Agradecimientos

En nombre de quienes realizamos este trabajo queremos agradecer a nuestras familias por el apoyo incondicional durante todo nuestro proceso educativo, gracias por acompañarnos durante esta etapa de nuestra vida que ya está por culminar.

A la Universidad Tecnológica de Pereira y al programa de Ingeniería de Sistemas y sus docentes que nos brindaron el apoyo necesario para terminar nuestras carreras.

Y finalmente al ingeniero Guillermo Solarte por la mentoría y acompañamiento durante el desarrollo de este proyecto.

Tabla de Contenido

Lista de tablas

Lista de figuras

CAPITULO I

GENERALIDADES

1.1.Título del proyecto

Sistema inteligente para la asignación de horarios a los docentes de la Tecnología en Desarrollo de Software.

1.2.Introducción

Los problemas de asignación de horarios y recursos en las instituciones educativas han sido objeto de estudio por años, estos problemas varían en su alcance y complejidad dependiendo de los requerimientos de la institución. Es importante encontrar soluciones óptimas que agilicen los procesos operacionales de asignación de profesores, aulas y cursos puesto que muchos modelos tradicionales no ofrecen soluciones satisfactorias y significan para las instituciones demoras y costos adicionales.

El problema de asignación de aulas en una institución académica según Carter y Tovey (1992) se refiere a la asignación de clases, que se reúnen en diferentes períodos de tiempo, a la vez que respetan una serie de restricciones y preferencias operativas. Ellos también discutieron su complejidad computacional dependiendo si el problema es intervalo que se refiere a cuando las clases sólo se dan una vez a la semana o sin intervalo cuando las clases se pueden dar más de una vez a la semana, para este proyecto trabajaremos entonces con un problema de asignación de salones sin intervalo.

En consecuencia, el presente ejercicio investigativo pretende describir y aplicar metodologías basadas en un modelo matemático, y específicamente, la combinación de la teoría

de investigación de operaciones, la inteligencia artificial y la resolución del modelo obtenido a través del lenguaje de programación Python, por sus herramientas de desarrollo en materia de ciencias de datos, para permitir la optimización de las operaciones de asignación de horarios y salones de clase, de manera que no resulten cruces de ningún tipo, y se supere la extenuante asignación a través de métodos tradicionales.

1.3.Planteamiento del problema

El problema de asignación de salones de manera que no haya cruces en los horarios concierne a la investigación de operaciones, más específicamente son problemas de combinatoria en los que se busca un objeto en una colección finita de objetos y típicamente la cantidad de éstos crece exponencialmente (Cook, Cunningham & Pulleyblank, 2003).

Se requiere un modelo matemático que de manera óptima asigne salones a los diferentes grupos de las materias del programa de Tecnología de Desarrollo de Software de la Universidad Tecnológica de Pereira con las restricciones de que no se puede asignar el mismo salón para dos o más materias en el mismo bloque horario y no se le puede asignar a un profesor un grupo en un horario que no tenga disponible.

Para el programa de Tecnología en Desarrollo de Software de la Universidad Tecnológica de Pereira es de mucha importancia que el proceso de asignación de horarios de sus docentes sea lo más rápido y fluido posible para que sus operaciones de matrícula semestrales no se vean retrasadas y así puedan ofrecer una educación de calidad; y uno de los problemas inmediatos que requiere solución es el cruce salones que sucede cuando se le asignan a materias diferentes la misma aula en el mismo bloque horario.

1.4.Justificación

Desde hace ya bastante tiempo la manera como se maneja la información ha ido cambiando y casi ha obligado a los directivos de toda institución a automatizar los procesos y a optimizar la forma en que se manipula la información. Por consiguiente, en esta era ya es casi una necesidad el hecho de actualizarse constantemente para suprimir técnicas y procesos obsoletos que pueden llegar a retrasar determinadas actividades en una organización particular impidiendo así que puedan ser competitivas frente a otras.

Para las Directivas del Programa de Tecnología de Desarrollo de Software, el proceso de asignación de la carga académica se ha convertido en una tarea complicada debido a los diversos cambios que se ven obligados a realizar para poder llevar a cabo con éxito dicho proceso y teniendo en cuenta que se realiza manualmente.

De ahí surge entonces la necesidad de implementar un sistema informático que a través del uso de algoritmos genéticos sirva como herramienta de apoyo a los Directivos del programa para que puedan desempeñar con éxito esta tediosa tarea.

El problema de la asignación de la carga académica no solo le compete al Programa de Tecnología de Desarrollo de Software, sino que abarca todos los programas de la Universidad Tecnológica de Pereira, y más aún a los programas
nuevos que sufren más cambios en su recurso profesoral, viéndose esto directamente reflejado en un mayor trabajo por parte de los responsables de este proceso en cada facultad. El alcance de este proyecto está limitado al programa de Tecnología en Desarrollo de Software, pero puede extenderse en cualquier momento a otras facultades ajustándose a sus requerimientos.

1.5.Objetivos

1.5.1. Objetivo general.

Diseñar e implementar un software a partir de un modelo matemático que permita optimizar la asignación de salones del programa Tecnología en Desarrollo de Software de la Universidad Tecnológica de Pereira.

1.5.2. Objetivos específicos.

- Indagar y recopilar la información necesaria para optimizar la asignación de horarios mediante la aplicación de un modelo matemático.
- Consultar los modelos matemáticos y sus técnicas de solución para la optimización de la asignación de horarios.
- Proponer un modelo matemático
- Escribir y depurar el código en el lenguaje de programación Python.
- Diseñar e implementar el sistema inteligente a partir del algoritmo del modelo matemático.
- Realizar pruebas y simulaciones de escenarios en los que el programa sería utilizado y evaluar su desempeño.

1.5.3. Antecedentes de la solución al problema

Existen algunas aplicaciones y soluciones previas enfocadas a la resolución de problemas de asignación de horarios las cuales sirvieron de base para implementar y mejorar las funcionalidades de software adaptándolo a las necesidades específicas del programa de Tecnología de Desarrollo de Software. Si bien estas aplicaciones presentan funcionalidades que o se adaptan o no están enfocados a suplir completamente las necesidades que tiene el programa de Tecnología de Desarrollo de Software de la Universidad. Es por eso que se quiso implementar este aplicativo utilizando metodologías y conceptos que permitieran brindar una solución factible al problema a abordar.

CAPÍTULO II

ESTADO DEL ARTE

2.1. Trabajos anteriores

En el amplio mundo de la asignación de horarios existen dos conceptos fundamentales los cuales contienen las investigaciones abordadas con respecto a la asignación de carga docente y su distribución en los distintos espacios estos conceptos son *timetabling* y *scheduling*. Un problema de *scheduling* es aquel que se trata de la asignación de recursos en el tiempo para llevar a cabo un conjunto de tareas (Baker,1974) y tiene como objetivo minimizar el costo total de los recursos asignados (Wren 1996).

Se define como *timetabling* como el problema de asignar ciertos recursos, sujeto a limitaciones, en un número limitado de horarios y lugares físicos con el objeto de satisfacer una serie de objetivos en el mayor grado posible (Wren 1996).

Los problemas de *timetabling* contienen restricciones fuertes y restricciones débiles. (Larrosa 2003) Las restricciones fuertes o duras son aquellas que deben ser satisfechas de forma mandatoria y su no cumplimiento inhabilita la solución. Corresponden generalmente a restricciones de tipo espacial o temporal que deben cumplirse. Algunos ejemplos de restricciones fuertes son: no se puede usar simultáneamente un espacio por dos grupos diferentes, no se puede asignar una persona simultáneamente en dos lugares diferentes. Las restricciones débiles o suaves son aquellas que es deseable que se cumplan, pero que su incumplimiento no inhabilita la solución, pero la hace de menor calidad. Generalmente son restricciones de preferencia o de priorización y muchas veces estas restricciones son las que se desean maximizar (o minimizar según sea el caso) para buscar acercarse a la solución óptima.

Así, el objetivo en un problema de timetabling es minimizar el incumplimiento de las restricciones suaves (Tallabó 1999).

Si bien cada problema de asignación de horarios es diferente a otro, existe una clasificación básica de estos que facilita su comprensión (Schaerf 1995)

- Programación de clases y profesores: Considera la asignación de un conjunto de docentes a un conjunto de cursos en un período de tiempo. Este tipo de problema puede considerarse un problema de tipo scheduling.
- Programación de clases y salas: Considera la asignación de un conjunto de cursos a un conjunto de salas en un período de tiempo. Este tipo de problema puede considerarse un problema de tipo timetabling.
- Asignación de horarios de exámenes: Consiste en la calendarización de los exámenes de los alumnos, asignando docentes y salas. Este tipo de problemas son más simples debido a que los exámenes no suponen una persistencia en el tiempo, si no se aplican solo una vez en el período.

Generalmente, los problemas de asignación de horarios académicos corresponden a una suma de los dos primeros tipos: Programación de clases y profesores y Programación de clases y salas.

Por ende, generalmente los problemas de asignación de horarios tienen componentes de timetabling y de scheduling. Por esto mismo, muchas veces el problema se aborda en dos etapas (Granada 2006), asignando en una primera etapa los docentes a los cursos en un horario y en una segunda etapa la combinación docente-curso a la sala.

Por consiguiente la asignación de horarios es un problema que en su temática general es universal a todas las instituciones de educación, pero las restricciones particulares hacen que cada instancia requiera una solución propia.

CAPITULO III

MARCO TEÓRICO

3.1. Algoritmos genéticos

Los algoritmos genéticos son aquellos que basan su funcionamiento en mecanismos de selección natural y supervivencia de los individuos más aptos, simulando los procesos de evolución postulados por Darwin. "*los algoritmos genéticos son algoritmos de búsqueda basados en la mecánica de selección natural y de la genética natural. Combinan la supervivencia del más apto entre estructuras de secuencias con un intercambio de información estructurado, aunque aleatorizado, para constituir así un algoritmo de búsqueda que tenga algo de las genialidades de las búsquedas humanas*" [Goldberg 1989].

Por consiguiente, los algoritmos genéticos los cuales también son definidos como algoritmos evolutivos se fundamentan en individuos los cuales son representación de la solución al problema y las bases de esta solución están presentes al existir operaciones que permiten el intercambio de información entre distintos individuos para así lograr una transformación permitiendo a la adaptabilidad en el ambiente.

Los individuos considerados como aptos son los seleccionados para reproducirse, generando nuevos individuos como mejores capacidades asimilando soluciones más aptas. Este proceso de evolución se repite de hasta que se acerque a una solución considerada como la solución más optima. En cada repetición de un algoritmo genético bien diseñado se da una mejor solución a la presentada en la repetición anterior, de igual manera aquellos individuos que no se

adecuen correctamente a la situación, tendrán menos probabilidades de continuar en el proceso reproductivo, impidiendo así que en una repetición las posibles soluciones se degraden.

El uso de los algoritmos genéticos supone grandes ventajas en comparación con otros algoritmos ya que estos además de ser aplicables a una gran cantidad de problemas proporcionan soluciones muy aceptables en tiempos razonables haciéndolo competir con otros métodos de optimización. Una muestra de ello es que los algoritmos genéticos han sido muy usados en los problemas de timetabling o scheduling generando unas soluciones bastante cercanas a las óptimas puesto que estos algoritmos permiten la adaptabilidad a condiciones específicas de cada problema. La Figura 1 muestra el proceso general que sigue un algoritmo genético en su ejecución.

En los algoritmos evolutivos existen operaciones como las de selección, cruza y mutación las cuales son aplicadas a los cromosomas donde la operación de selección es la encargada de elegir los mejores cromosomas para su proceso de reproducción, cruza es la encargada de aprobar el proceso de reproducción entre dos cromosomas y finalmente la operación de mutación es aquella que permite transformar un cromosoma en alguna de sus aristas con tal de explorar espacios de soluciones que serían imposibles con la cruza.

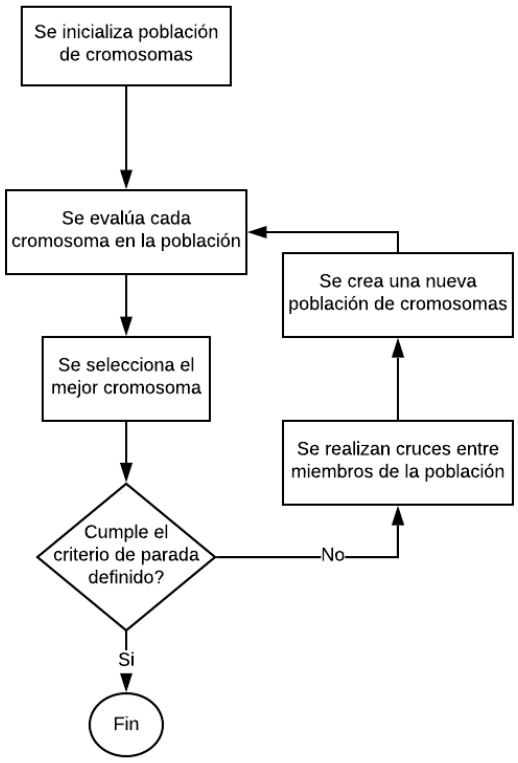


Figura 1. Proceso de ejecución general de un algoritmo genético

3.2. Cromosoma

En los algoritmos evolutivos un cromosoma se asimila como una representación de una solución al problema. El cromosoma contiene la información relevante del problema representando una estructura de datos la cual se encuentra compuesta por uno o más genes. Además, la configuración definida para el cromosoma es de gran importancia ya que esta garantizará la rapidez en la ejecución del algoritmo donde la función objetivo pueda determinar si el cromosoma es apto o no.

3.3. Cruce de cromosomas

En cada nueva generación sucesiva en la ejecución del algoritmo genético, una porción de la población es elegida para reproducirse entre sí de manera que se obtenga una nueva generación mejor, esta evaluación se hace por medio del cálculo de una función de aptitud o *fitness* la cual mide la calidad de la solución representada. La Figura 2 muestra cómo se hace el proceso de evaluación de la población, se representa la población con un modelo binario y se hace el cruce de los mejores candidatos para finalmente dar resultado a una nueva generación con un puntaje de aptitud más alto.

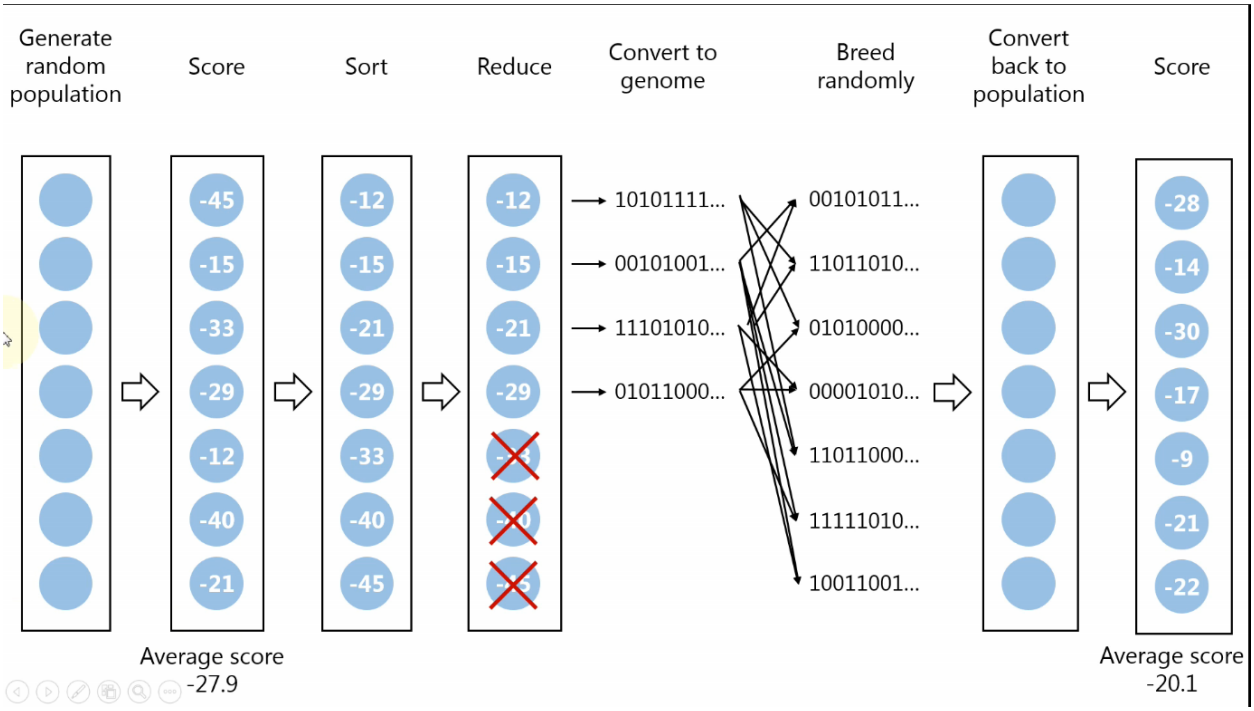


Figura 2. Proceso de evaluación y cruce población en un algoritmo genético

Fuente: [svinec]. (2018, Enero 31). Operation Scheduling Using Genetic Algorithm in Python [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=e84aLKGWtW4&list=LLXmIOs0E2vot0cKq5gMx97Q&index=2&t=640s

3.4. Método de selección por torneo

La selección de torneo es una estrategia de selección utilizada para seleccionar los candidatos más aptos de la generación actual en un algoritmo genético. Estos candidatos seleccionados se pasan a la próxima generación. En una selección de torneo *K-way*, seleccionamos k-individuos y realizamos un torneo entre ellos. Solo se elige el candidato más apto entre los candidatos seleccionados y se pasa a la próxima generación. De esta manera, se llevan a cabo *n* torneos y tenemos nuestra selección final de candidatos que pasarán a la próxima generación. También tiene un parámetro llamado presión de selección, que es una medida probabilística que significa la probabilidad de participación de un candidato en un torneo. Si el tamaño del torneo es mayor, los candidatos débiles tienen menos posibilidades de ser seleccionados, ya que tiene que competir con más candidatos y es más probable que tenga que competir con candidatos muy fuertes o aptos de la población.

El parámetro de presión de selección determina la tasa de convergencia del algoritmo genético. A más presión de selección mayor será la tasa de convergencia. Los algoritmos genéticos pueden identificar soluciones óptimas o casi óptimas en una amplia gama de presiones de selección. La selección de torneo también funciona para valores negativos de aptitud. La Figura 3 muestra la lógica del proceso de selección por torneo.

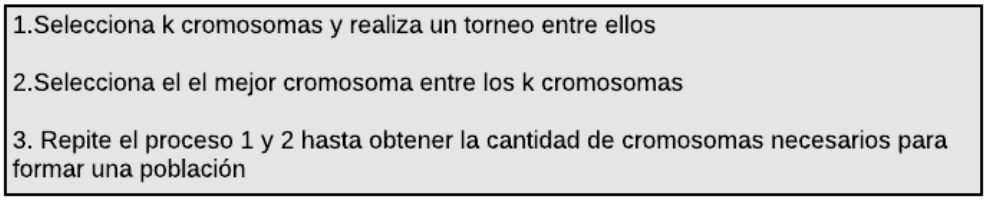


Figura 3. Algoritmo de selección por torneo

3.5. Lenguaje de Modelado Unificado (UML)

El Lenguaje de Modelamiento Unificado es un lenguaje gráfico el cual fue creado para para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software cuyo principal objetivo es brindar a arquitectos de sistemas, ingenieros y desarrolladores de software las herramientas para el análisis, el diseño y la implementación de sistemas basados en software, así como para el modelado de procesos de negocios y similares.

3.5.1 Diagrama de clases

Este tipo de diagramas nos sirven para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de contención además de ofrecer algunos beneficios como ilustrar modelos de datos para sistemas de información, sin importar qué tan simples o complejos sean, comprender mejor la visión general de los esquemas de una aplicación, expresar visualmente cualesquier necesidades específicas de un sistema y divulgar esa información en toda la empresa, crear diagramas detallados que resaltan cualquier código específico que será necesario programar e implementar en la estructura descrita, ofrecer una descripción independiente de la implementación sobre los tipos empleados en un sistema que son posteriormente transferidos entre sus componentes.

Los diagramas de clases se encuentran compuestos por dos tipos de elementos y estos son las **clases** y **relaciones** donde las clases son la unidad básica que encapsula toda la información correspondiente a determinado objeto permitiendo así modelar el entorno de estudio como lo puede ser una asignatura. La representación gráfica de una clase es un rectángulo el cual se divide en tres secciones.

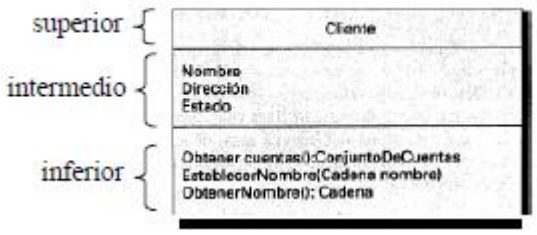


Figura 4. Representación diagrama de clases

Fuente: Pressman, Roger S. Ingeniería del Software: Un enfoque práctico. Quinta edición. España. Mc Graw Hill.

En la parte superior se encuentra el nombre de la clase seguido de la parte intermedia la cual contiene los atributos que caracterizan a esa clase siendo estas de tipo público, privado o protegido, de igual manera en la parte inferior se encuentran los métodos y operaciones las cuales determinan la forma como va a comunicar el objeto con su entorno.

Un ejemplo de un diagrama de clases es la representación de un sistema administrativo hotelero donde se puede mostrar las relaciones entre cada objeto incluidas la información de huéspedes, las responsabilidades del personal y la ocupación por habitación.

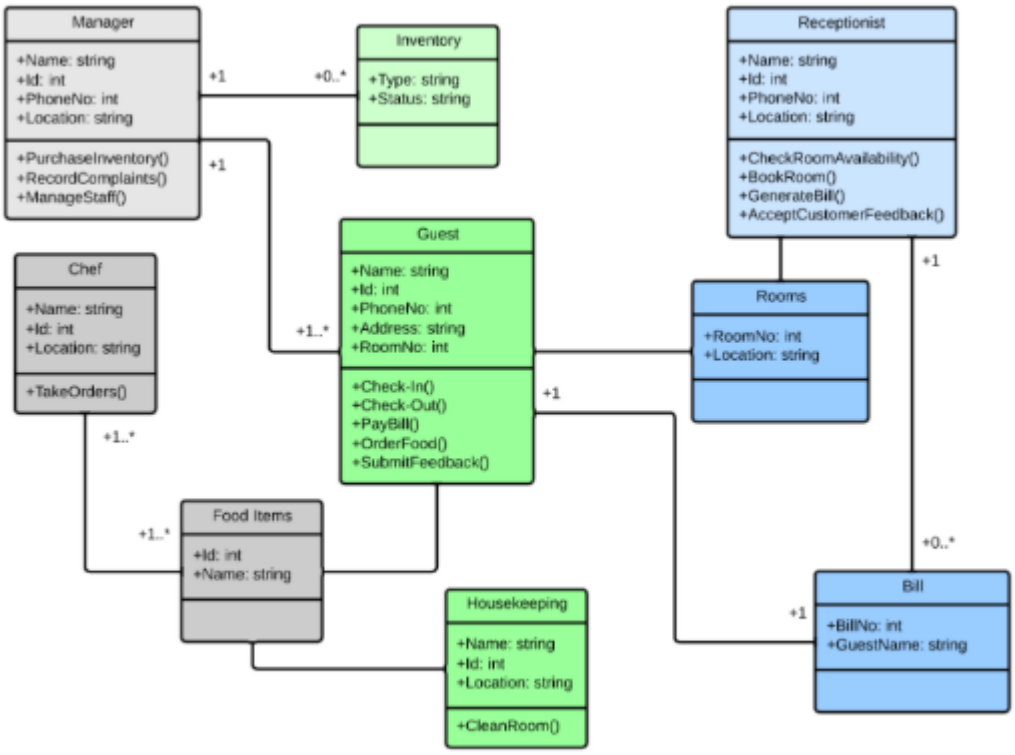


Figura 5. Ejemplo diagrama de clases.

Fuente: Ejemplos de diagramas de clase. (2019, Noviembre 27). Recuperado de https://www.lucidchart.com/pages/es/tutorial-de-diagrama-de-clases-uml#section_3

3.5.2 Diagrama de componente

El objetivo de los diagramas de componentes es el dar una descripción clara de los elementos que constituyen a nivel físico el sistema y las relaciones con el mismo aportando simplicidad incluso a los procesos más complejos. Los componentes de estos diagramas son una representación de todos los tipos de software que representan las aplicaciones informáticas desde archivos hasta paquetes.

Estos diagramas muestran las opciones de realización incluyendo el código fuente, binario y ejecutable,de igual manera un diagrama de componentes está constituido por interfaces, relaciones de dependencia, generalización, asociación y realización. Un ejemplo de aplicación de los diagramas de componentes es el diseño de un sistema para la gestión de bibliotecas donde actualmente los sistemas de biblioteca organizan todo tipo de datos ingresados y retirados por los usuarios. Estas operaciones crean una red de relaciones entre los componentes del sistema de biblioteca las cuales pueden ser examinadas en la siguiente ilustración.

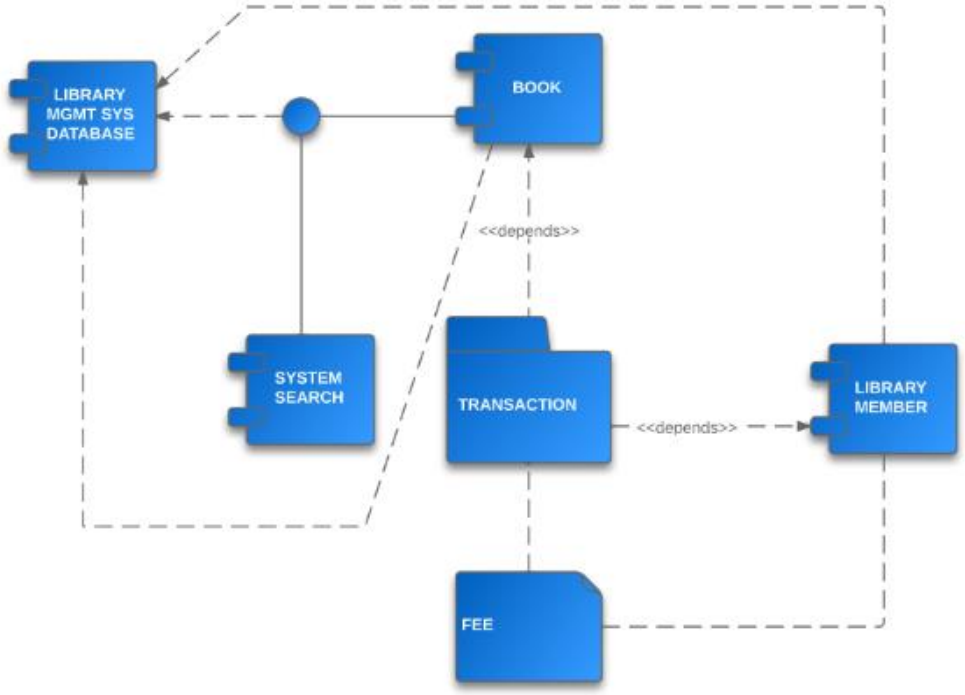


Figura 6. Ejemplo diagrama de componentes.

Fuente: Ejemplos de diagramas de componentes. (2019, Noviembre 27). Recuperado de https://www.lucidchart.com/pages/uml-component-diagram#section_5

3.5.3 Diagrama relacional

Este modelo se obtiene en tiempo de diseño de la base de datos. Fue propuesto por el Doctor en Ciencias de la Computación y Matemáticas Aplicadas Peter Chen en 1976 y desde entonces se viene utilizando de una forma muy global. Se caracteriza por utilizar una serie de símbolos y reglas para representar los datos y sus relaciones. Con los diagramas relacionales conseguimos representar de manera gráfica la estructura lógica de una base de datos.

Los elementos de un diagrama relacional son las entidades, las relaciones y los atributos donde la entidad se trata de un objeto del que se recoge información de interés de cara a la base de datos. Gráficamente se representan mediante un rectángulo. Dentro de las entidades pueden ser fuertes o débiles. Las fuertes son las que no dependen de otras entidades para existir, mientras que las entidades débiles siempre dependen de otra entidad sino no tienen sentido por ellas mismas y las relaciones se definen como la relación como una asociación de dos o más entidades.

A cada relación se le asigna un nombre para poder distinguirla de las demás y saber su función dentro del modelo entidad-relación. Otra característica es el grado de relación, siendo las de grado 1 relaciones que sólo relacionan una entidad consigo misma. Las de grado 2 son relaciones que asocian dos entidades distintas, y las de grado n que se tratan de relaciones que unen más de dos entidades. Las relaciones se representan gráficamente con rombos, dentro de ellas se coloca el nombre de la relación. Finalmente, los atributos se definen como cada una de las propiedades de una entidad o relación. Cada atributo tiene un nombre y todos los posibles valores que puede tener. Dentro de una entidad tiene que haber un atributo principal que identifica a la entidad y su valor tiene que ser único.

Un ejemplo de un diagrama relacional es el sistema de una biblioteca cuya representación se presenta a continuación.

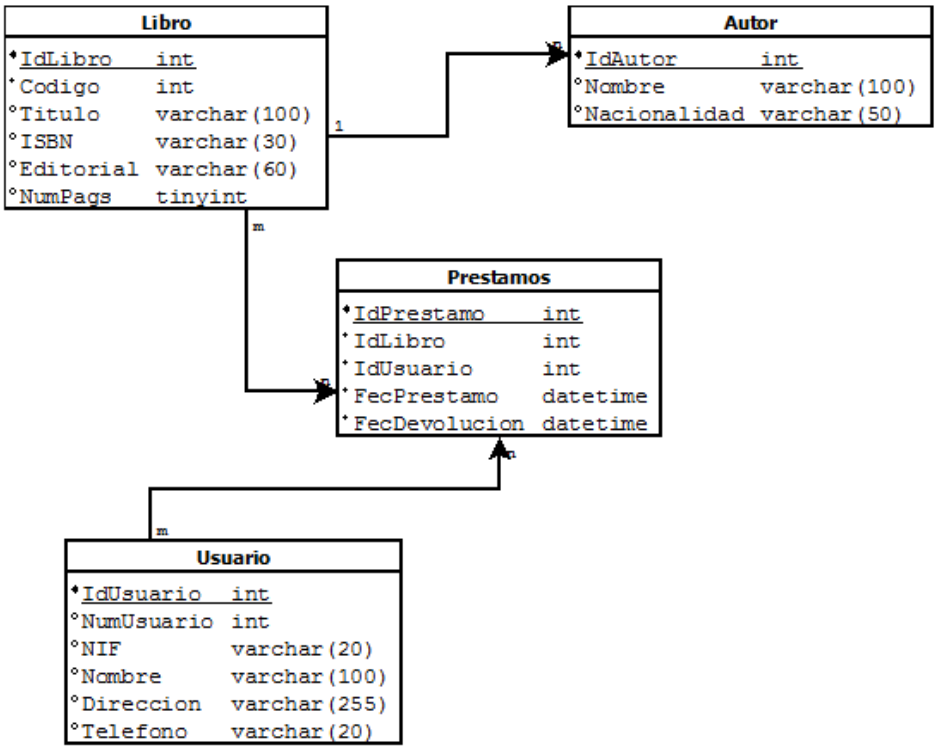


Figura 7. Ejemplo modelo relacional.

Fuente: Diseño de Base de Datos, Modelo Relacional. (2019, Noviembre 27). Recuperado de https://soloesciencia.com/2017/08/04/911/

3.5.4 Diagrama de paquetes

Los diagramas de paquetes nos permiten organizar los elementos modelados con UML, facilitandonos de esta forma el manejo de los modelos de un sistema complejo, además estos diagramas definen un espacio de nombres donde pueden coexistir dos elementos UML con el mismo nombre pero si estos se encuentran en paquetes distintos. En estos diagramas los paquetes pueden ser simples estructuras conceptuales o también pueden estar reflejados en la

implementación, también permiten dividir un modelo para agrupar y encapsular sus elementos en unidades lógicas individuales.

Los diagramas son de gran utilidad puesto que se pueden utilizar para plantear la arquitectura del sistema a nivel macro donde los paquetes pueden estar anidados unos dentro de otros y unos paquetes pueden depender de otros paquetes. Un ejemplo donde se pueden aplicar los diagramas de componentes es en modelo de negocio de comidas rápidas.

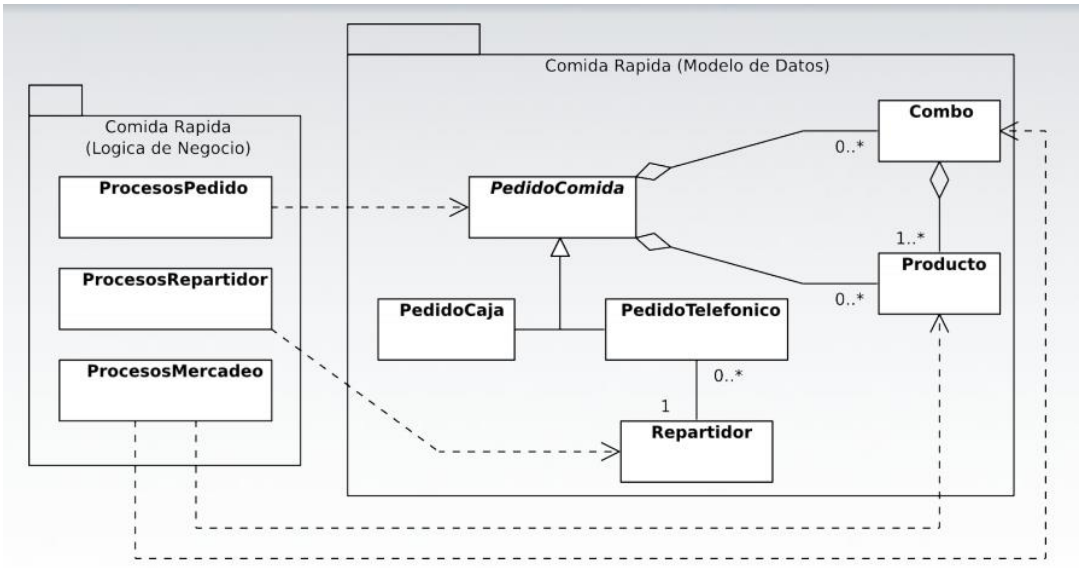


Figura 8. Ejemplo diagrama de paquetes.

Fuente: UML Diagramas de paquetes (UML ilustrado) (2009, Septiembre). Recuperado de codecompiling.net/files/slides/UML_clase_05_UML_paquetes.pdf

CAPITULO IV

METODOLOGÍA

4.1. Identificación de requerimientos para el programa de tecnología en desarrollo de software

El diseño e implementación del software descrito en este trabajo se hará para el programa de tecnología en desarrollo de software de la Universidad Tecnológica de Pereira el cual cuenta con 6 semestres académicos, en cada semestre el pensum cuenta con los cursos que se muestran en la Tabla 1 junto con sus respectivas intensidades horarias por semana.

Tabla 1

Pensum académico por semestres de la Tecnología en Desarrollo de Software de la UTP

Código	**Nombre**	**HT**	**HP**	**HE**	**HC**
	Semestre I				
CB115	Matemáticas Básicas	4	2	14	5
TS125	Programación I	4	2	12	5
BU101	Deportes I		2	4	1
BA172	Humanidades I	2		6	2
TS293	Introducción a la Informática	2	2	8	3
TS372	Desarrollo del Pensamiento Lógico	2		6	2
Total		16	8	50	18
	Semestre II				
TS283	Matemáticas Discreta	4	2	12	5
TS225	Programación II	4	2	12	5
CB223	Álgebra Lineal	3	1	8	3
BU201	Deportes II		2	4	1
TS252	Técnicas de la Comunicación Hablada y Escrita	2		4	2
Total		13	7	40	16

Código	Nombre	HT	HP	HE	HC
	Semestre III				
TS104	Estructura de Datos	4	2	10	4
TS323	Lógica	4		8	3
TS433	Programación Orientada a Objetos	2	2	8	3
TS343	Técnicas de Desarrollo de Software	2	2	8	3
TS253	Emprendimiento	2		4	3
Total		14	6	38	16
	Semestre IV				
TS416	Entornos de Desarrollo de Software		6	12	4
TS424	Bases de Datos	2	2	12	4
TS434	Ingeniería de Software	4	2	12	4
TS442	Estadística	4		8	2
TS454	Fundamentos de Redes de Computadores	2	2	8	3
Total		12	12	52	17
	Semestre V				
TS514	Laboratorio de Ingeniería de Software	3	3	12	4
TS523	Sistemas Operativos	2	2	8	3
TS534	Programación Web	2	4	12	4
TS543	Electiva I	2	2	8	3
TS563	Preparación Trabajo de Grado	2	2	10	4
Total		11	13	50	18
	Semestre VI				
TS612	Constitución Política	2		4	2
TS624	Administración y Planeación de Proyectos de Software	3	1	8	4
TS633	Administración de Sistemas	2	2	8	3
TS643	Electiva I		4	8	3
TS653	Electiva II		4	8	3
TS664	Trabajo de Grado	2	2	10	4
Total		9	13	46	19

Nota. HT y HP son Horas Teórica y Horas Prácticas, la suma de éstas da como resultado la intensidad horaria semanal de la materia para ser usada en las restricciones del algoritmo. HE y HC son Horas de Estudio y Horas Cátedra, serán ignoradas puesto que no son pertinentes al problema abordado en este trabajo.

4.1.2. Restricciones y condiciones para la asignación de horarios.

Se deben definir las condiciones que deben cumplir los horarios al momento de ser asignados para el caso de la tecnología en desarrollo de software y las restricciones que serán la base de evaluación del diseño del algoritmo genético, para todas las características dadas de cursos, grupos, salones, profesores y bloques horarios. Las Figuras 9 describen específicamente cuáles son esas condiciones que se deben cumplir y la Figura 10 describe las restricciones que debe cumplir una solución preliminar encontrada por el algoritmo genético para ser aceptada como solución definitiva.

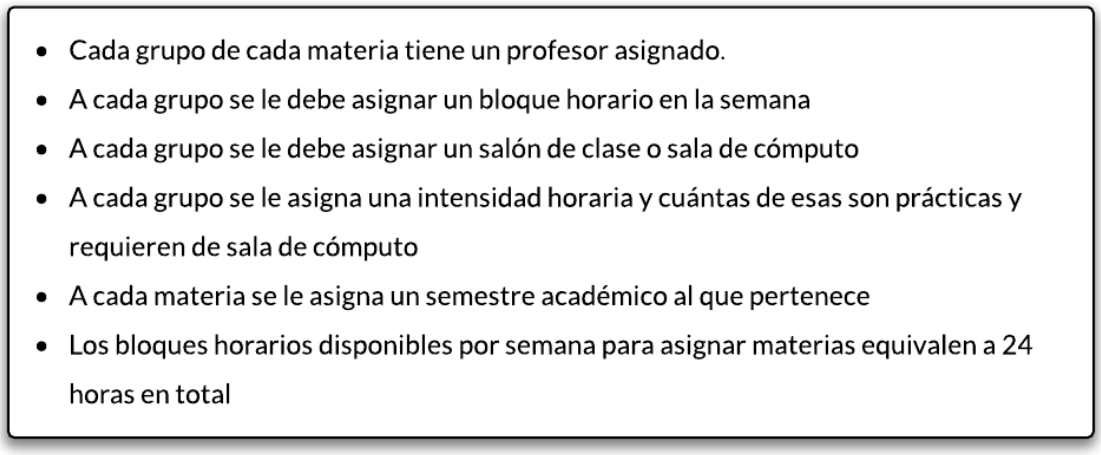


Figura 9. Condiciones para asignación de horario

- R01: Si dos grupos diferentes tienen clase en el mismo bloque horario, no pueden tener el mismo profesor asignado.
- R:02 Si dos grupos diferentes tienen clase a la misma hora, no se les puede asignar el mismo salón de clase o sala de cómputo.
- R03: Si dos materias diferentes pertenecen al mismo semestre semestre académico no pueden ser dadas en el mismo bloque horario aún si se cumplen todas las restricciones anteriores.
- R04: Si una materia requiere *n* cantidad de bloques horarios en sala debe tener *n* cantidad de bloques con sala asignada, (restricción rígida)

Figura 10. Restricciones para la aceptación de un horario asignado

4.2. Diseño del algoritmo genético

4.2.1. Diseño de cromosomas.

El cromosoma es un factor importante en el algoritmo genético, en el cual el diseño de los cromosomas afectará la implementación de los cruces que se hagan para cada generación. La representación de cada cromosoma será una lista de listas, cada lista equivale a un bloque horario asignado, y cada lista dentro de la lista de bloques contiene cada una de las condiciones mencionadas en la Figura 10 organizadas de la siguiente manera:

Ls[] = {Semestre, ID_curso, #_de_salon, profesor, ID_bloque_horario, día}

La inicialización del algoritmo se hace mediante la asignación aleatoria de valores *n* veces hasta obtener todos los cursos para obtener el primer cromosoma y luego se repite esta

operación *k* veces hasta obtener el tamaño de población deseada, la Figura 11 describe la lógica de este proceso. Los valores aleatorios son obtenidos de una base de datos local donde estarán previamente guardados los datos de los cursos, profesores, salones y horarios.

```
While i<n:
    ls_ls.append(ls)
    While j<k:
        ls_ls[i][j] = random()
        j = j+i
    i = i+1
```

Figura 11. Proceso de inicialización de cromosomas

4.2.2. Diseño del cruce y función de aptitud

La aptitud o *fitness* es un valor numérico obtenido del resultado de la evaluación de las restricciones definidas para el caso estudiado, por cada restricción que no se cumpla en un cromosoma un valor de conflictos *conflicts* aumenta en 1, este valor es pasado a la función de *fitness*, cuyo valor máximo es de 1.0 cuando no hay ningún conflicto.

$$fitness = \frac{1}{(conflicts + 1)}$$

Para el cruce de cromosomas en la población de cada generación se utiliza el método de selección por torneo para, inicialmente, elegir los miembros que se cruzarán. Posteriormente se procede a cruzar los dos miembros que sobrevivieron el torneo intercambiando información guardada en sus listas, se evalúa la aptitud del cromosoma hijo resultante y si éste tiene un valor de *fitness* más alto, reemplaza al miembro más débil de la generación. El proceso se repite por *x* generaciones hasta que uno de los cromosomas obtenga un valor de *fitness* de 1.0.

4.3. Modelo de implementación

4.3.1. Diagrama de clases.

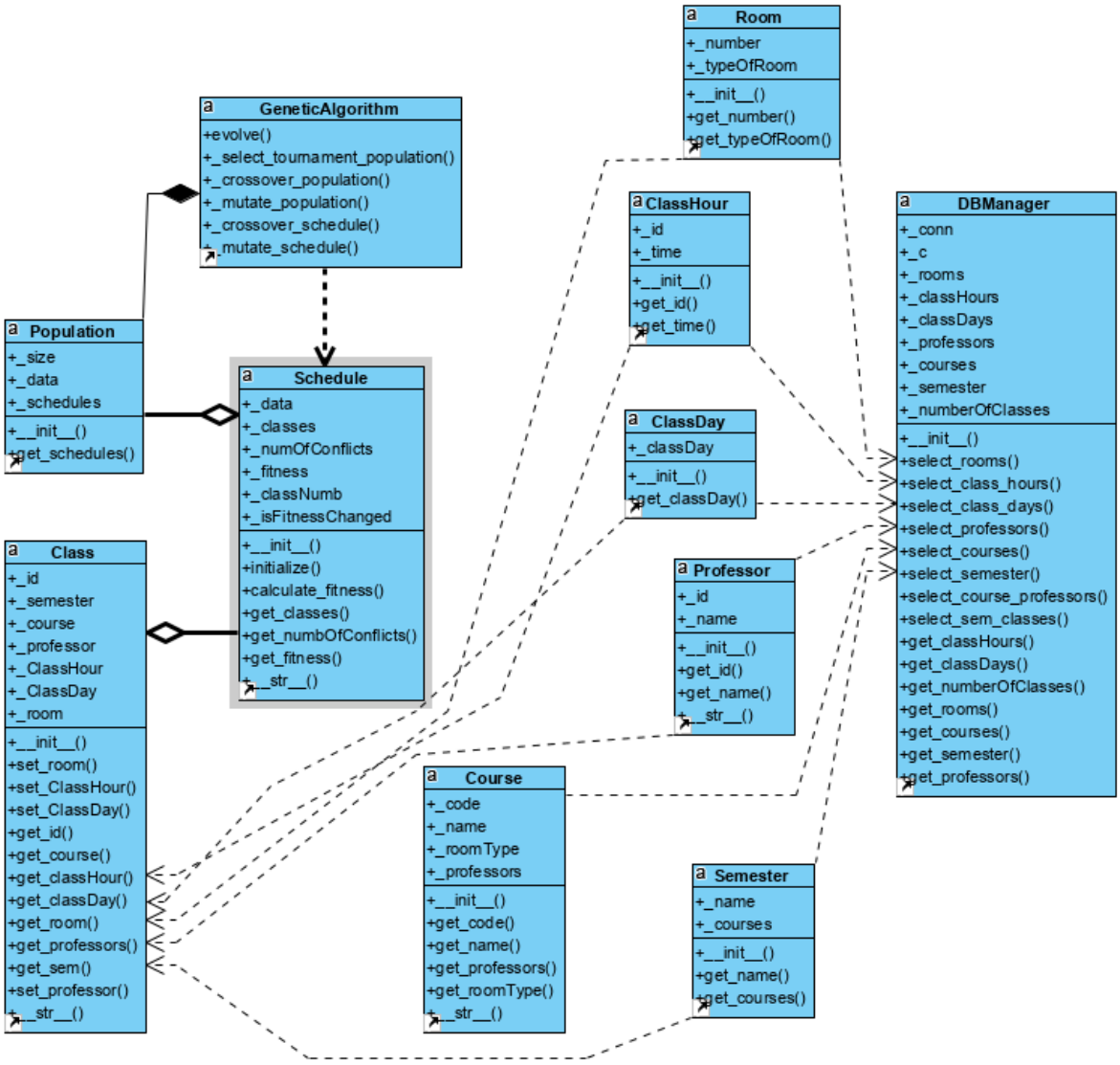


Figura 12. Diagrama de clases

Tabla 2

Clase DBManager

ID:	C001
Nombre de Clase:	DBManager
Atributos	
rooms	Salones de clase disponibles
classHours	Horas de clase semanales
professors	Profesores disponibles
courses	Materias del pensum académico
semester	Lista de semestres académicos
Métodos	
select_	Métodos de recuperación de datos de la Base de Datos
get_	Constructores de datos recuperados

Tabla 3

Clase Room

ID:	C002
Nombre de Clase:	Room
Atributos	
number	Identificador del salón
typeOfRoom	Tipo de salón (1 = Salón, 2 = Sala de Cómputo)
Métodos	
get_	Constructores de datos de salones recuperados

Tabla 4

Clase ClassHour

ID:	C003
Nombre de Clase:	ClassHour
Atributos	
id	Identificador del bloque horario
time	Rango de horas del bloque
Métodos	
get_	Constructores de datos de bloque horario recuperados

Tabla 5

Clase ClassDay

ID:	C004
Nombre de Clase:	ClassDay
Atributos	
classDay	Día de clase (L, M, MI, J, V, S)
Métodos	
get_	Constructores de datos de días de clase

Tabla 6

Clase Professor

ID:	C005
Nombre de Clase:	Professor
Atributos	
id	Identificador del profesor

name	Nombre del profesor
Métodos	
get_	Constructores de datos de profesores

Tabla 7

Clase Semester

ID:	C006
Nombre de Clase:	Semester
Atributos	
name	Nombre del semestre
courses	Lista de cursos que pertenecen al semestre
Métodos	
get_	Constructores de datos de semestres académicos

Tabla 8

Clase Course

ID:	C007
Nombre de Clase:	Course
Atributos	
code	Identificador del curso
name	Nombre completo del curso
roomType	Tipo de salón que requiere (sala de computo o salón)
professors	Id del profesor que dicta el curso
Métodos	
get_	Constructores de datos de cursos

Tabla 9

Clase Class

ID:	C008
Nombre de Clase:	Class
Atributos	
id	Identificador de clase
Hereda los siguientes atributos: Semester:name , Course:code, Professor:id, ClassHour:id, ClassDay:classDay, Room:number	
Métodos	
get_	Constructores de datos de salones recuperados
set_	Asigna los atributos a la instancia de la Clase

Tabla 10

Clase Schedule

ID:	C009
Nombre de Clase:	Schedule
Atributos	
data	Estructura vacía para acomodar población
classes	Las clases de cada bloque horario que se asignarán
numberOfConflicts	Número de conflictos encontrados en la generación
isFitnessChanged	Booleano para evaluar si fitness ha cambiado
Métodos	
get_	Constructores de datos del horario obtenido
initialize	Inicializa una población con valores aleatorios
calculate_fitness	Calcula la aptitud de la población

Tabla 11

Clase GeneticAlgorithm

ID:	C010
Nombre de Clase:	Room
Atributos	
Métodos	
evolve	Constructores de datos de salones recuperados
select_tournament_population	Selecciona un miembro de la población que se pasa como argumento por medio de método de selección por torneo
crossover_population	Realiza un cruce entre dos cromosomas de la población
mutate_population	Utiliza MUTATION_RATE para reducir la posibilidad de que el algoritmo se quede en cromosomas que no mejorarán
crossover_schedule	Realiza un cruce entre dos asignaciones horarias dentro de un cromosoma
mutate_schedule	Utiliza MUTATION_RATE para reducir la posibilidad de que el algoritmo se quede en asignaciones horarias que no mejorarán

4.3.2. Diagrama relacional.

En la Figura 13 se describe el modelo de la base de datos utilizada en la implementación:

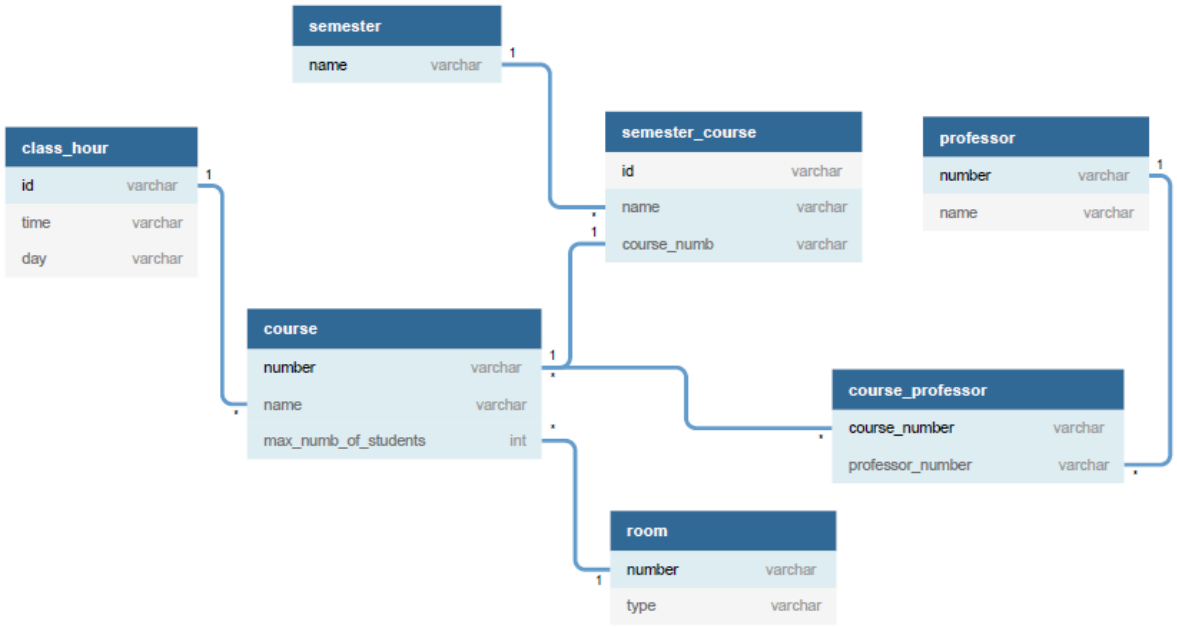


Figura 13. Diagrama relacional de la base de datos

La base de datos utilizada en la implementación es estática para cada ejecución del software y debe permanecer así, puesto que de lo contrario el orden de introducción de los datos afectaría negativamente el tiempo de ejecución del algoritmo al perder la capacidad de inicializarse con las suposiciones (correctas) de asignación de profesores y asignación de salones o salas de cómputo.

4.3.3. Diagrama de componentes.

La Figura 14 muestra los componentes del sistema:

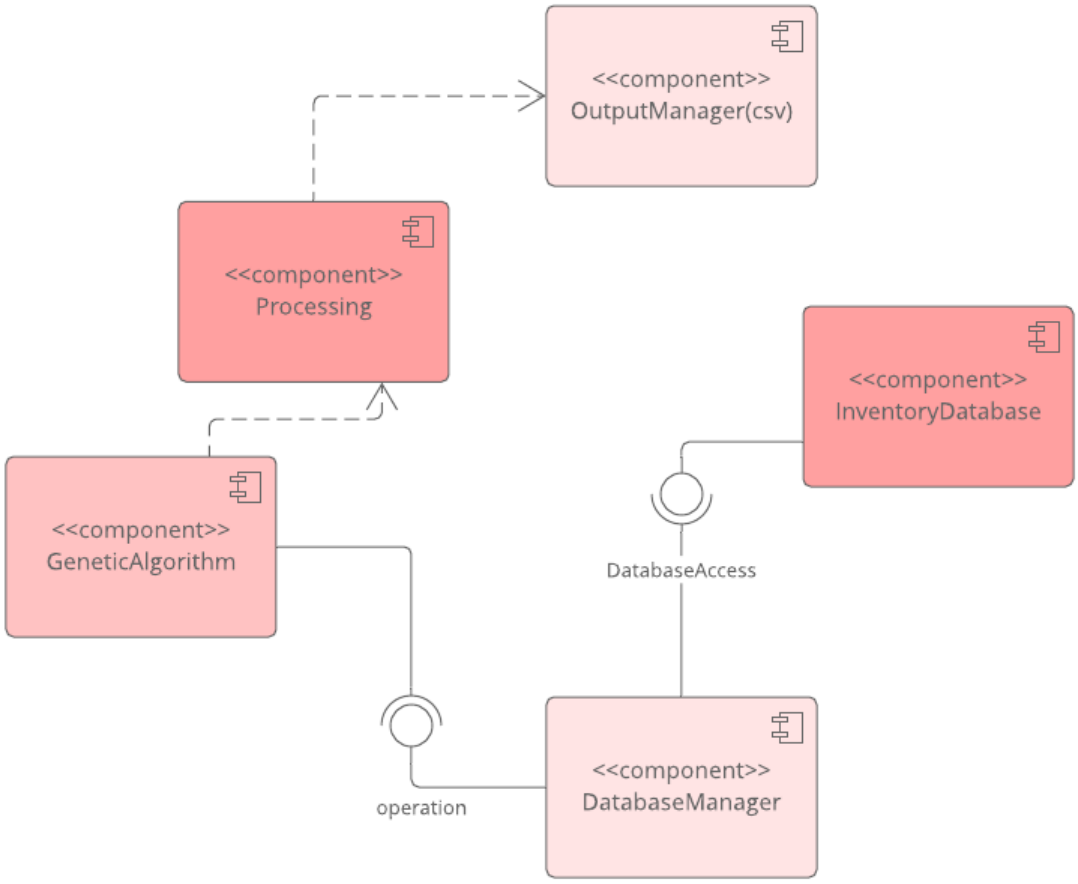


Figura 14. Diagrama de componentes

- InventoryDatabase: Información de la base de datos pre generada.

- DatabaseManager: Sistema encargado de recuperar la información de la base de datos y crear las Clases necesarias.
- GeneticAlgorithm: Implementación del algoritmo genético que asigna los bloques horarios de cada semestre académico asegurando encontrar una solución con cero conflictos y que por tanto satisfaga la función de *fitness* definida.
- Processing: Encargado de traducir el genoma obtenido del algoritmo genético en una lista de cadenas de texto para generar una salida legible.
- OutputManager: Organiza las salidas procesadas en archivos .csv, uno por cada semestre, con los horarios respectivos.

4.3.4. Herramientas de la implementación.

La implementación del sistema fue realizada en el lenguaje de programación Python versión 3.8, Python es conocido por su código conciso y legible, y es casi inigualable en lo que respecta a la facilidad de uso y la simplicidad, particularmente para los nuevos desarrolladores. Esto tiene varias ventajas para el aprendizaje de máquina y el aprendizaje profundo.

Tanto el aprendizaje de máquina como el aprendizaje profundo se basan en algoritmos extremadamente complejos y flujos de trabajo de múltiples etapas, por lo que cuanto menos tenga que preocuparse un desarrollador por las complejidades de la codificación, más podrá concentrarse en encontrar soluciones a los problemas y alcanzar los objetivos del proyecto.

La sintaxis simple de Python significa que también es más rápido en desarrollo que muchos lenguajes de programación, y permite al desarrollador hacer pruebas ágilmente durante la implementación. Además, el código fácil de leer es invaluable para la codificación

colaborativa, o cuando los proyectos de aprendizaje automático o aprendizaje profundo cambian de manos entre los equipos de desarrollo. Esto es particularmente cierto si un proyecto contiene una gran cantidad de lógica empresarial personalizada o componentes de terceros, como es el caso particular de este trabajo puesto que el sistema implementado podría utilizarse a futuro en la Tecnología de Desarrollo de Software para la página web del programa.

Para la implementación de la base de datos se utilizó una herramienta simple para la creación de bases de datos locales llamada SQLite, la cual es una librería interna de Python 3.8. SQLite genera un archivo llamado UTP_TDS_TimeTable.db persistente por medio de un script que crea las tablas e inserta la información de cursos, semestres, bloques horarios, salones, salas y profesores, la Figura 15 muestra las tablas que se crean en el archivo.

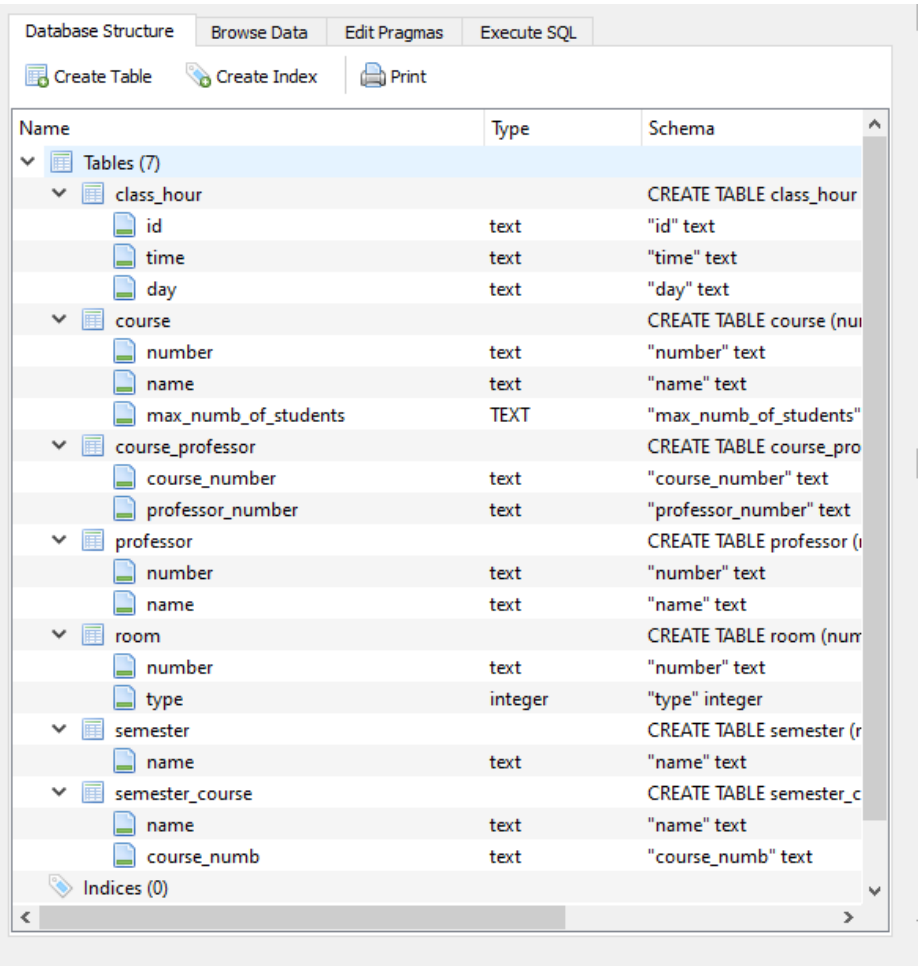


Figura 15. Tablas con los datos iniciales del programa

Finalmente se utiliza la librería *csv*, también incluida en Python 3.8, para recoger los datos del cromosoma que cumple con la función de aptitud (definida en la sección 4.2.2.) y darles formato de tabla donde las cabeceras serán horas y días de la semana disponibles para dar clases del programa y cada celda contiene: nombre de la materia que se dará en esas horas, el profesor correspondiente y el número identificador del salón asignado. La Tabla 12 muestra el formato de las salidas del sistema.

Tabla 12

Formato de salidas del sistema de asignación de horarios de la Tecnología en Desarrollo de Software

Hora	**Lunes**	**Martes**	**Miércoles**	**Jueves**	**Viernes**	**Sábado**
08:00 – 10:00	-	-	-	-	-	*Asignación*
10:00 – 12:00	-	-	-	-	-	*Asignación*
18:00 – 20:00	*Asignación*	*Asignación*	*Asignación*	*Asignación*	*Asignación*	-
20:00 – 22:00	*Asignación*	*Asignación*	*Asignación*	*Asignación*	*Asignación*	-

Nota. *Asignación* es la tupla de {materia, profesor, salón}

CAPITULO V

RESULTADOS

5.1. Ejecución del algoritmo genético

La ejecución del algoritmo genético es secuencial, lo que significa que el tiempo de finalización depende únicamente de la capacidad del procesador de la máquina en la que se ejecutó para realizar operaciones en un solo núcleo. La razón de ello es que es necesario terminar por completo el cruce de población de una generación para poder continuar a evaluar la siguiente generación.

Para probar el algoritmo implementado se realizaron pruebas incrementales en complejidad agregando un semestre (y todos los datos que este contiene) en cada ejecución subsecuente del programa.

Tabla 13

Contenido de datos de cada prueba

Prueba 1	{SEMESTRE I}
Prueba 2	{SEMESTRE I, SEMESTRE II}
Prueba 3	{SEMESTRE I, SEMESTRE II, SEMESTRE III}
Prueba 4	{SEMESTRE I, SEMESTRE II, SEMESTRE III, SEMESTRE IV}
Prueba 5	{SEMESTRE I, SEMESTRE II, SEMESTRE III, SEMESTRE IV, SEMESTRE V}
Prueba 6	{SEMESTRE I, SEMESTRE II, SEMESTRE III, SEMESTRE IV, SEMESTRE VI}

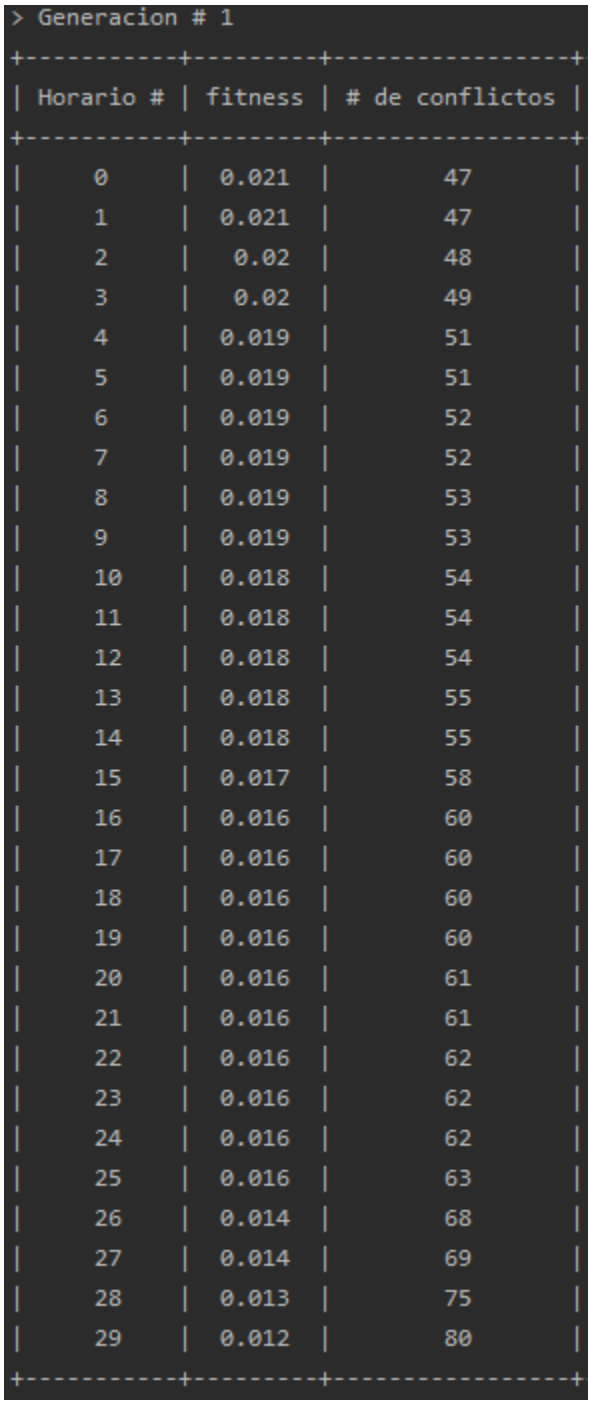
La complejidad del problema es de $O(n^n)$ y, efectivamente, se puede evidenciar un incremento exponencial en la cantidad de generaciones necesarias para encontrar la solución cada vez que se agrega un semestre académico a la base de datos.

Tabla 14

Número de generaciones que le tomó al algoritmo encontrar una solución por cada prueba

Prueba nº	**Generaciones**	**Tiempo de Ejecución (ms)**
Prueba 1	2	401
Prueba 2	21	2515
Prueba 3	88	7815
Prueba 4	324	32554
Prueba 5	964	107219
Prueba 6	5149	693213

A continuación, se muestra la aptitud o *fitness* de las diferentes generaciones durante el proceso evolutivo del algoritmo tomando como muestra la Prueba 5. La Figura 16 muestra una generación inmadura, la Figura 17 muestra una generación intermedia y la Figura 18 muestra la generación final que encontró la solución correcta. Se elige la Prueba 5 porque si lo hacemos con una más compleja, la generación intermedia siempre tendrá 1 conflicto, resolver este conflicto es un paso que termina llevando miles de generaciones en el caso de, por ejemplo, la Prueba 6.



> Generacion # 1

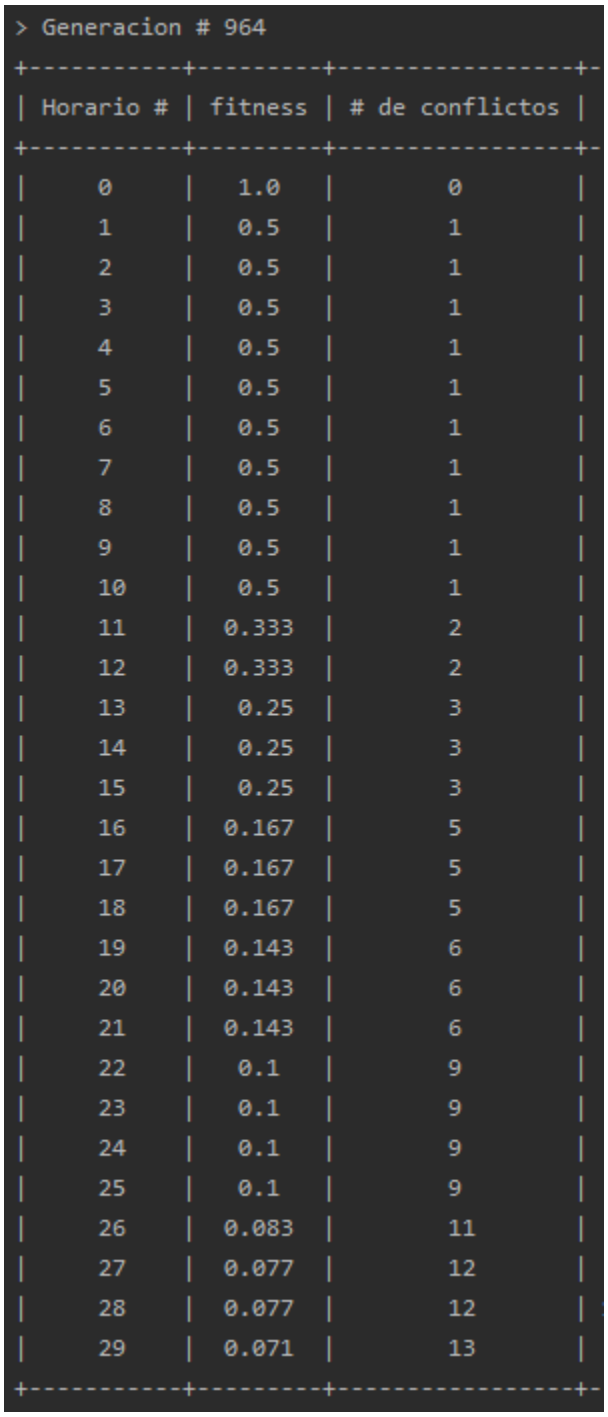
Horario #	fitness	# de conflictos
0	0.021	47
1	0.021	47
2	0.02	48
3	0.02	49
4	0.019	51
5	0.019	51
6	0.019	52
7	0.019	52
8	0.019	53
9	0.019	53
10	0.018	54
11	0.018	54
12	0.018	54
13	0.018	55
14	0.018	55
15	0.017	58
16	0.016	60
17	0.016	60
18	0.016	60
19	0.016	60
20	0.016	61
21	0.016	61
22	0.016	62
23	0.016	62
24	0.016	62
25	0.016	63
26	0.014	68
27	0.014	69
28	0.013	75
29	0.012	80

Figura 16. Generación inmadura

> Generacion # 266

Horario #	fitness	# de conflictos
0	0.25	3
1	0.25	3
2	0.25	3
3	0.25	3
4	0.25	3
5	0.25	3
6	0.25	3
7	0.25	3
8	0.25	3
9	0.25	3
10	0.25	3
11	0.2	4
12	0.167	5
13	0.167	5
14	0.167	5
15	0.143	6
16	0.143	6
17	0.143	6
18	0.125	7
19	0.125	7
20	0.125	7
21	0.125	7
22	0.1	9
23	0.1	9
24	0.091	10
25	0.091	10
26	0.091	10
27	0.077	12
28	0.067	14
29	0.062	15

Figura 17. Generación intermedia



> Generacion # 964

Horario #	fitness	# de conflictos
0	1.0	0
1	0.5	1
2	0.5	1
3	0.5	1
4	0.5	1
5	0.5	1
6	0.5	1
7	0.5	1
8	0.5	1
9	0.5	1
10	0.5	1
11	0.333	2
12	0.333	2
13	0.25	3
14	0.25	3
15	0.25	3
16	0.167	5
17	0.167	5
18	0.167	5
19	0.143	6
20	0.143	6
21	0.143	6
22	0.1	9
23	0.1	9
24	0.1	9
25	0.1	9
26	0.083	11
27	0.077	12
28	0.077	12
29	0.071	13

Figura 18. Generación final

5.2. Salidas del sistema

Las salidas del sistema son generadas con un script que crea un archivo csv por cada semestre académico de la base de datos en el directorio donde se encuentren los archivos de la implementación, la Figura 19 muestra los archivos creados.

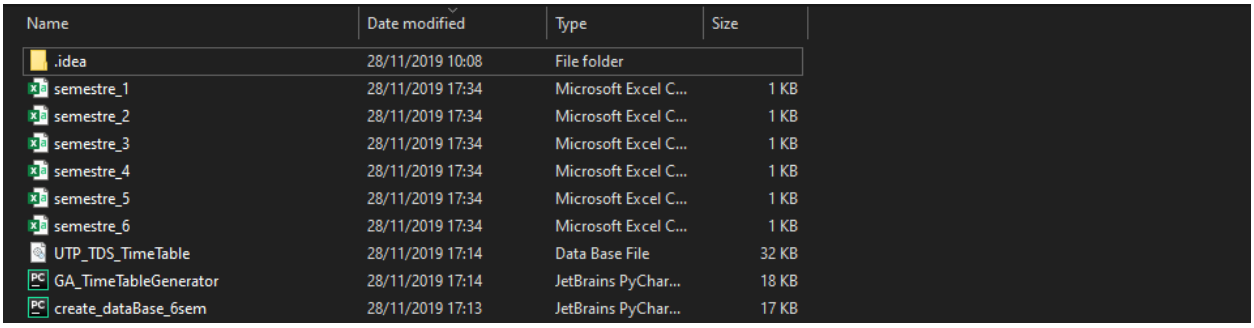


Figura 19. Archivos generados como salidas

Cada archivo csv contiene una tabla con la información del horario asignado a cada semestre del programa de Tecnología en Desarrollo de software las Tablas 14 a 19 a muestran los horarios asignados por el sistema para los seis semestres del programa.

Tabla 15

Horario asignado al Semestre I

Horas	Lunes	Martes	Miercoles	Jueves	Viernes	Sabado
08:00 - 10:00						Desarrollo_Pensamiento, Andres Restrepo, R17
10:00 - 12:00						Programacion_I, Guillermo Toro, R9
18:00 - 20:00	Deportes_I, Diana Patricia Jurado, R13	Matematica_I, Hugo Humberto Morales, R17	Introduccion_Informatica, Andres Restrepo, R5	Humanidades_I, Diana Patricia Jurado, R8	Introduccion_Informatica, Andres Restrepo, L1	
20:00 - 22:00	Programacion_I, Guillermo Toro, L4		Matematica_I, Hugo Humberto Morales, R13	Matematica_I, Hugo Humberto Morales, R9	Programacion_I, Guillermo Toro, L1	

Tabla 16

Horario asignado al Semestre II

Horas	Lunes	Martes	Miercoles	Jueves	Viernes	Sabado
08:00 - 10:00						Tecnicas_Comunicacion, Diana Patricia Jurado, R15
10:00 - 12:00						Matematica_Discreta, Hugo Humberto Morales, R17
18:00 - 20:00		Programacion_II, Guillermo Toro, R7	Matematica_Discreta, Hugo Humberto Morales, R9	Programacion_II, Guillermo Toro, L7	Programacion_II, Guillermo Toro, L7	
20:00 - 22:00	Matematica_Discreta, Hugo Humberto Morales, R17	Algebra_Lineal, Hugo Humberto Morales, R9	Deportes_II, Diana Patricia Jurado, R10		Algebra_Lineal, Hugo Humberto Morales, R2	

Tabla 17

Horario asignado al Semestre III

Horas	Lunes	Martes	Miercoles	Jueves	Viernes	Sabado
08:00 - 10:00						Programacion_Orientada_a_Objetos, Angel Augusto Agudelo, L5
10:00 - 12:00						Programacion_Orientada_a_Objetos, Angel Augusto Agudelo, R7
18:00 - 20:00	Estructura_de_Datos, Angel Augusto Agudelo, R18	Logica, Carlos Andres Calvo, R5	Logica, Carlos Andres Calvo, R17	Tecnicas_de_Desarrollo de Software, Carlos Andres Calvo, R1	Tecnicas_de_Desarrollo de Software, Carlos Andres Calvo, L6	
20:00 - 22:00	Estructura_de_Datos, Angel Augusto Agudelo, L6		Estructura_de_Datos, Angel Augusto Agudelo, R9		Emprendimiento, Guillermo Roberto Solarte, R9	

Tabla 18

Horario asignado al Semestre IV

Horas	Lunes	Martes	Miercoles	Jueves	Viernes	Sabado
08:00 - 10:00						Entornos_de_Desarrollo_de_Software, Guillermo Roberto Solarte, L7
10:00 - 12:00						Estadistica, Ana Maria Lopez, R5
18:00 - 20:00	Ingenieria_de_Software, Guillermo Roberto Solarte, R12	Estadistica, Ana Maria Lopez, R2	Entornos_de_Desarrollo_de_Software, Guillermo Roberto Solarte, L2	Bases_de_Datos, Angel Augusto Agudelo, R18	Fundamentos_de_Redes, Ana Maria Lopez, R16	
20:00 - 22:00	Ingenieria_de_Software, Guillermo Roberto Solarte, L1	Entornos_de_Desarrollo_de_Software, Guillermo Roberto Solarte, L2	Fundamentos_de_Redes, Ana Maria Lopez, L3	Ingenieria_de_Software, Guillermo Roberto Solarte, L5	Bases_de_Datos, Angel Augusto Agudelo, L7	

Tabla 19

Horario asignado al Semestre V

Horas	Lunes	Martes	Miercoles	Jueves	Viernes	Sabado
08:00 - 10:00						Laboratorio_Ingenieria_Software, Ana Maria Lopez, L1
10:00 - 12:00						Programacion_Web, Saulo Torres, L6
18:00 - 20:00	Sistemas_Operativos, Juan de Jesus Veloza, L5	Programacion_Web, Saulo Torres, R18	Electiva_I, Juan de Jesus Veloza, R11	Programacion_Web, Saulo Torres, L3	Sistemas_Operativos, Juan de Jesus Veloza, R14	
20:00 - 22:00	Laboratorio_Ingenieria_Software, Ana Maria Lopez, L2	Prep_Trabajo_de_Grado, Saulo Torres, R1	Electiva_I, Juan de Jesus Veloza, L1	Prep_Trabajo_de_Grado, Saulo Torres, R3	Laboratorio_Ingenieria_Software, Ana Maria Lopez, R8	

Tabla 20

Horario asignado al Semestre VI

Horas	Lunes	Martes	Miercoles	Jueves	Viernes	Sabado
08:00 - 10:00						Constitucion_Politica, Juan de Jesus Veloza, R1
10:00 - 12:00						
18:00 - 20:00	Electiva_I, Guillermo Valencia, L3	Trabajo_de_Grado, Guillermo Valencia, R4	Admon_Planeacion_Proyectos_Software, Guillermo Valencia, R18	Admon_Planeacion_Proyectos_Software, Guillermo Valencia, R15	Admon_de_Sistemas, Alexis Montoya, L2	
20:00 - 22:00	Electiva_II, Alexis Montoya, R9	Trabajo_de_Grado, Guillermo Valencia, R6	Admon_de_Sistemas, Alexis Montoya, R1	Electiva_I, Guillermo Valencia, R2	Electiva_II, Alexis Montoya, L2	

5.3. Conclusiones

- La inteligencia artificial permite encontrar solución a problemas extremadamente complejos y que con heurísticas de bajo nivel no pueden ser resueltos o encontrar una solución en un tiempo razonable.
- El problema de asignación de horarios es un problema de complejidad computacional muy alta. Encontrar soluciones eficientes está en el interés de quienes necesitan realizar estas operaciones, ya sea en colegios, universidades o cualquier otra institución educativa con un número grande de estudiantes, profesores y aulas.
- Entre más suposiciones se puedan hacer en un algoritmo genético, menos tiempo le toma encontrar una solución. Esto se evidenció al momento de programar las restricciones de salones, cuando se decidió convertirla en una restricción rígida (de manera que sólo se

asignaran salones a las clases que requerían salones e igual para las salas de cómputo), la cantidad de generaciones necesarias para encontrar un horario sin conflictos se redujo considerablemente.

- Debido a su código conciso y legible, y es casi inigualable en lo que respecta a la facilidad de uso y la simplicidad, Python es un lenguaje de programación excelente para la implementación de algoritmos de inteligencia artificial complejos puesto que permite concentrarse en encontrar soluciones a los problemas y alcanzar los objetivos del proyecto, en lugar de distraerse con las complejidades de la codificación

Bibliografía

Alzate, P. M. (2017). Modelo de Programación Entera para la Asignación de Actividades Académicas Optimizando Espacios en Aulas de Clase. Universidad Tecnológica de Pereira. 56-70.

Andrews, S. Fastqc, (2010). A quality control tool for high throughput sequence data.

Augen, J. (2004). Bioinformatics in the post-genomic era: Genome, transcriptome, proteome, and information-based medicine. Addison-Wesley Professional.

Baker K. R.; Martin, J. B. (1974). An experimental comparison of solution algorithms for the single-machine tardiness problem. Naval Research Logistics, vol. 21.

Blankenberg, D., Kuster, G. V., Coraor, N., Ananda, G., Lazarus, R., Mangan, M., ... & Taylor, J. (2010). Galaxy: a web-based genome analysis tool for experimentalists. Current protocols in molecular biology, 19-10.

Boizumault, P.; Delon, Y.; PERIDY, L. (1996). Constraint Logic Programming for Examination Timetabling. The journal of logic programming. 217-233.

Bolger, A., & Giorgi, F. Trimmomatic: A Flexible Read Trimming Tool for Illumina NGS Data. URL http://www. usadellab. org/cms/index. php.

Cook, W. J.; Cunningham, W. H.; Pulleyblank, W. R.; Schrijver, A. (2003). Combinatorial Optimization. Ed. Springer, vol A, 1-2.

Carter, M. W. (1989). A Lagrangian Relaxation Approach To The Classroom Assignment Problem. Information Processing. 230-245

Carter, M. W.; Tovey, C. A.(1992). When is the Classroom Assignment Problem Hard? Operations Research, vol. 40, 28-41.

Dammak, A.; Elloumi, A.; Kamoun, H. (2006). Classroom Assignment for Exam Timetabling. Advances in Engineering Software, vol. 37, no. 10, 659-666 .

Giardine, B., Riemer, C., Hardison, R. C., Burhans, R., Elnitski, L., Shah, P., ... & Nekrutenko, A. (2005). Galaxy: a platform for interactive large-scale genome analysis. Genome research, 15(10), 1451-1455.

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Adison- Wesley Publishing Company.

López, C. A., Caicedo, S. M. Algoritmo hiperheurístico para generar una solución factible al problema de la programación de docentes en una institución de educación media para una jornada escolar específica. Universidad Tecnológica de Pereira.

Sapna, S., Tamilarasi, Kumar, M. (2012). Implementation of Genetic Algorithm in Predicting Diabetes. International Journal of Computer Science, Vol 9.

Satia, G., Gudani, K., Wibowo, D. (2011). Genetic Algorithm for Scheduling Courses. Petra Christian University.

Wren, A. (1992). Scheduling, timetabling and rostering — A special relationship? Lecture Notes in Computer Science.